©SNAK, 2009

Inter J Nav Archit Oc Engng (2009) 1:89~94
http://dx.doi.org/10.2478/IJNAOE-2013-0011

OpenFOAM: Open source CFD in research and industry

Hrvoje Jasak[1,2]

[1]Wikki Ltd. London, United Kingdom, [2]FSB, University of Zagreb, Croatia

ABSTRACT: *The current focus of development in industrial Computational Fluid Dynamics (CFD) is integration of CFD into Computer-Aided product development, geometrical optimisation, robust design and similar. On the other hand, in CFD research aims to extend the boundaries of practical engineering use in "non-traditional" areas. Requirements of computational flexibility and code integration are contradictory: a change of coding paradigm, with object orientation, library components, equation mimicking is proposed as a way forward. This paper describes OpenFOAM, a C++ object oriented library for Computational Continuum Mechanics (CCM) developed by the author. Efficient and flexible implementation of complex physical models is achieved by mimicking the form of partial differential equation in software, with code functionality provided in library form. Open Source deployment and development model allows the user to achieve desired versatility in physical modeling without the sacrifice of complex geometry support and execution efficiency.*

KEY WORDS: CFD; Open Source; Finite volume; Object-Oriented; C++; Equation mimicking.

INTRODUCTION

Leading simulation software in Computational Continuum Mechanics (CCM) combines accurate and robust numerics, complex geometry support and an impressive range of physical models in a user-friendly user environment. Current simulation challenges are related to integration and automation of simulation tools in a Computer Aided Engineering (CAE), including automatic geometry retrieval, surface and volume meshing, scripted code execution with variants of boundary conditions, material properties and model settings as well as sensitivity and optimisation studies. Here, dynamic mesh handling, parallel computing support and a wide range of pre-implemented physical models is considered as standard.

In research, the focus is shifted to efficient and reliable implementation of complex and coupled physical models, aimed to extend numerical modeling capabilities beyond current engineering practice. The objective of software design for "research use" is to allow the researcher to experiment with new physical models, validate them against experimental data and examine their performance on real industrial problems.

Two sets of requirements are sometimes contradictory: ease of implementation of new models does not necessarily go hand in hand with the needs of industrial environment. In an ideal world, transition from a model development framework to industrial application should be seamless: re-using the same software and validation data.

This paper describes the design of OpenFOAM, an object-oriented library for Computational Continuum Mechanics designed in pursuit of the above. In place of monolithic software design and "user coding" extensions, OpenFOAM implements the components of mesh handling, linear system and solver support, discretisation operators and physical models in library form, where they are re-used over a number of top-level solvers. Implementation of complex physical models follows the idea of mimicking the form of partial differential equations in software. Auxiliary tools, from pre-processing, mesh manipulation, data acquisition, dynamic mesh handling etc. are built into the system, bringing it to the level expected by industrial CFD tools.

OBJECT ORIENTATION AND EQUATION MIMICKING

Complexity of monolithic functional software stems from its data organisation model: globally accessible data is operated on by a set of functions. Here, each added feature interacts with all other parts of the code, potentially introducing new defects (bugs) – with the growing size of

Corresponding author: *Hrvoje Jasak*
e-mail: *h.jasak@wikki.co.uk and hrvoje.jasak@fsb.hr*

Copyright © 2009 Society of Naval Architects of Korea. Production and hosting by ELSEVIER B.V. This is an open access article under the CC BY-NC 3.0 license (http://creativecommons.org/licenses/by-nc/3.0/).

software, the data management and code validation problem necessarily grows out of control.

Object orientation attempts to resolve the complexity in a "divide and conquer" approach. The idea is to recognise self-contained objects in the problem and place parts of implementation into self contained types (classes) to be used in building the complexity. In C++, a class (object) consists of:

- *A public interface*, providing the capability to the user;
- *Private data*, needed to provide functionality and managed by the public interface.

As an example, consider a sparse matrix class. It will store matrix coefficients in its preferred manner (private data) and provide manipulation functions, e.g. matrix transpose, matrix algebra (addition, subtraction, multiplication by a scalar etc.). Each of these operates on private data in a controlled manner but its internal implementation details are formally independent of its interface.

Classes introduce new user-defined types into problem description, allowing the programmer to create a "look and feel" of the high-level code, ideally as close to the problem as possible.

In the arena of CCM, one can state that a natural language for physical model development already exists: it is a partial differential equation. Attempting to represent differential equations in their natural language in software as closely as possible is our stated goal.

Looking at the example of a turbulence kinetic energy equation in Reynolds Averaged Navier-Stokes (RANS) models:

$$\frac{\partial k}{\partial t} + \nabla\cdot(\mathbf{u}k) - \nabla\cdot\left[\left(\nu + \nu t\right)\nabla k\right] = \nu_t\left[\frac{1}{2}\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right)\right]^2 - \frac{\epsilon_0}{k_0}k \tag{1}$$

we shall follow the path to its encoded version in OpenFOAM:

```
solve
(
     fvm::ddt(k)
   + fvm::div(phi, k)
   - fvm::laplacian(nu() + nut, k)
  == nut*magSqr(symm(fvc::grad(U)))
   - fvm::Sp(epsilon/k, k)
);
```

Correspondence between Eqn. (1) and the code is clear, even with limited programming knowledge and without reference to object-orientation or C++.

FIVE BASIC CLASSES

The main objects used in code snippet above are listed below. Some basic types, like `scalar`, `vector`, `tensor`, `List`, `word` *etc.* underpin the system and will not be reviewed in detail.

Space and Time

In computational terms, the temporal dimension is split into a finite number of time-steps. Formally, it is sufficient to track the time step count and time increment Δt. A set of database operations associated with time-marching finds its natural home in the Time class, including simulation data output every n time-steps or x seconds of computational time and general time-related data handling, e.g. book-keeping for old-time level field data handling needed in temporal discretisation operators.

OpenFOAM implements *polyhedral mesh handling*, where a cell is described as a list of faces closing its volume, a face is an ordered list of point labels and points are gathered into an ordered list of (x, y, z) locations, stored as `vectors`. Lowlevel implementation is discretisation-independent, where the `polyMesh` class provides the addressing and mesh metrics (cell volumes, face areas, cell and face centres etc.). For convenient use with discretisation, basic mesh information is wrapped for convenience of use. `fvMesh`, for example, supports the Finite Volume Method (FVM), while `tetFemMesh` supports the Finite Element (FE) solver. In both cases, basic mesh structure and functionality is shared: a single mesh can simultaneously support the FVM and FEM solver without duplication of data and functionality.

Field Variable

Continuum mechanics operates on field variables, each of which is approximated as a list of typed values at pre-defined locations of the mesh. Thus, a `vectorField` class consists a list of vectors (three floating point numbers) and vector field operations: addition, subtraction, scalar multiplication, magnitude, dot- and cross-products etc. Arbitrary rank tensor fields are defined in the same manner.

Boundary conditions, encoded as patch fields carry *behaviour* in addition to its values. For example, a `fixedValue` field carries its values but shall not change on assignment: its value is fixed. Some other cases, like a `fixedGradient` field class can "evaluate" boundary values, given the internal field and a surface-normal gradient. This constitutes a family of related classes: each calculates its boundary value based on behaviour, but does the job in its own specific way.

Grouping the field data with its spatial dependence (reference to a mesh), boundary conditions and a dimension set creates a self contained *Geometric Field* object. Examples are the `volScalarField k` or `volVectorField U` in the code snippet above.

Matrix, Linear System and Linear Solver

A sparse matrix $[A]$ and linear system $[A][x] = [b]$ hold the result of discretisation and provide the machinery for its solution. It suffices to say that code organisation as presented above allows the FEM and FVM to share sparse matrix implementation and solver technology, resulting in considerable code re-use.

Discretisation Method

Discretisation operators assemble an implicit or explicit representation of operators, and are implemented in three levels.

Interpolation evaluates the field variable between computational points, based on prescribed spatial and temporal variation (shape function).

Differentiation, where calculus operations are performed on fields to create new fields. For example, the following code:

```
volVectorField gradP = fvc::grad(p);
```

creates a new FVM vector field of pressure gradient given a pressure field `p`. Calculus operator above carry the `fvc::` prefix.

Discretisation operates on differential operators (rate of change, convection, diffusion), and creates a discrete counterpart of the operator in sparse matrix form. Discretisation operators in software carry the `fvm::` prefix.

Physical Modelling Library

Taking object orientation further, one can recognise object families at the physics modelling level. For example, all RANS turbulence models in effect provide the same functionality: evaluating the Reynolds stress term $\overline{u'u'}$ in the momentum equation. Grouping them together guarantees inter-changeability and decouples their implementation from the rest of the flow solver. In such situation, the momentum equation communicates with a turbulence model through a pre-defined interface. A turbulence model contributes a discretisation matrix to the momentum equation, usually consisting of a diffusion term and explicit correction and no special knowledge of a particular turbulence model is needed.

Physics Solver

The components described so far act as a numerical tool-kit used to assemble various physics solvers. Each flow solver is a standalone tool, and handles only a narrow set of physics, eg. turbulent flow with LES, or partially premixed combustion. Capability of such solvers is underpinned by a combination of complex geometry support and parallelisation.

List of top-level solvers available in OpenFOAM closely mimics the capabilities of commercial CFD, with room for further vertical integration and customisation by the user.

OpenFOAM IN USE

In what follows, we shall illustrate the performance of top-level OpenFOAM solvers whose functionality is assembled from library components.

Flash-Boiling Model

In the spectrum of the flow with pressure-driven phase change, flash-boiling indicates the situation where the effect of inter-phase heat transfer plays a considerable role. At the cold end of the spectrum, cavitating flow models rely on the fact that low density of the cavitating vapour requires a small amount of energy transfer, allowing the use of equilibrium assumptions. In flash boiling, energy transfer is a limiting factor, and the phase equilibrium assumption no longer applies.

Under such conditions, the role of equation of state is replaced by a Homogeneous Relaxation Model (HRM), where the quality (mass fraction) x relaxes to equilibrium $\bar{x}$ over a time-scale Θ, obtained from empirical relations:

$$\frac{Dx}{Dt} = \frac{\bar{x} - x}{\Theta}, \tag{2}$$

$$\Theta = \Theta_{0\epsilon}^{\;-0.54}\phi^{1.76} \tag{3}$$

Other equations defining the system include conservation of mass:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\phi_v \rho) = 0 \tag{4}$$

and conservation of momentum:

$$\frac{(\partial \rho U^0)}{\partial t} + \nabla \cdot (\phi U^0) = -\nabla p^n + \nabla \cdot (\mu \nabla U^0). \tag{5}$$

Absence of the equation of state complicates the numerical implementation of this model. In recent work (Schmidt et al., 2009), Gopalakrishnan and Schmidt, of University of Massachusetts, Amherst develop a novel formulation of the pressure equation, encompassing the change of nature of the flow. The pressure equation reduces to its incompressible form in single phase flow and accounts for compressibility effects when $\frac{\partial\rho}{\partial p}$ is non-zero:

$$\frac{1}{\rho}\frac{(\partial\rho)}{\partial t}\bigg|_{x,h}\left(\frac{\partial(\rho p^{k+1})}{\partial t}\right)+\nabla\cdot\left(\rho U p^{k+1}\right)+\rho\nabla\cdot\phi^{*} \\ -\rho\nabla\frac{1}{a_p}\nabla p^{k+1}+M\left(p^{k}\right)+\frac{\partial M}{\partial p}\left(p^{k+1}-p^{k}\right)=0 \qquad (6)$$

Complexity of such algorithms is substantial, as implementation errors may appear in operator discretisation, boundary conditions, linearisation of coupling terms or in equation coupling. Isolating discretisation issues from equation coupling and model physics allows the researcher to concentrate on their area of expertise, while relying on correct operation of basic code components.

OpenFOAM, with its simple encoding of discretisation represents the above model in a concise manner. Efficiency of implementation, polyhedral mesh support, parallel processing capability *etc.* require no further work: they are embedded in the low-level code structure. Access to the source code at this level is not possible without the open source development paradigm and equation mimicking. As an illustration of code clarity, the complete flash boiling model code is listed below.

```
// Continuity equation
solve
(
    fvm::ddt(rho)
  + fvm::div(phiv, rho)
);


// Momentum equation
fvVectorMatrix UEqn
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - fvm::laplacian(mu, U)
);
solve(UEqn == -fvc::grad(p));


// Pressure equation
solve
(
    psi/sqr(rho)*(fvm::ddt(rho, p)
  + fvm::div(phi, p))
  + fvc::div(phivStar)
  - fvm::laplacian(rUA, p)
  + MSave + fvm::SuSp(dMdp, p)
  - dMdp*pSave
);
```

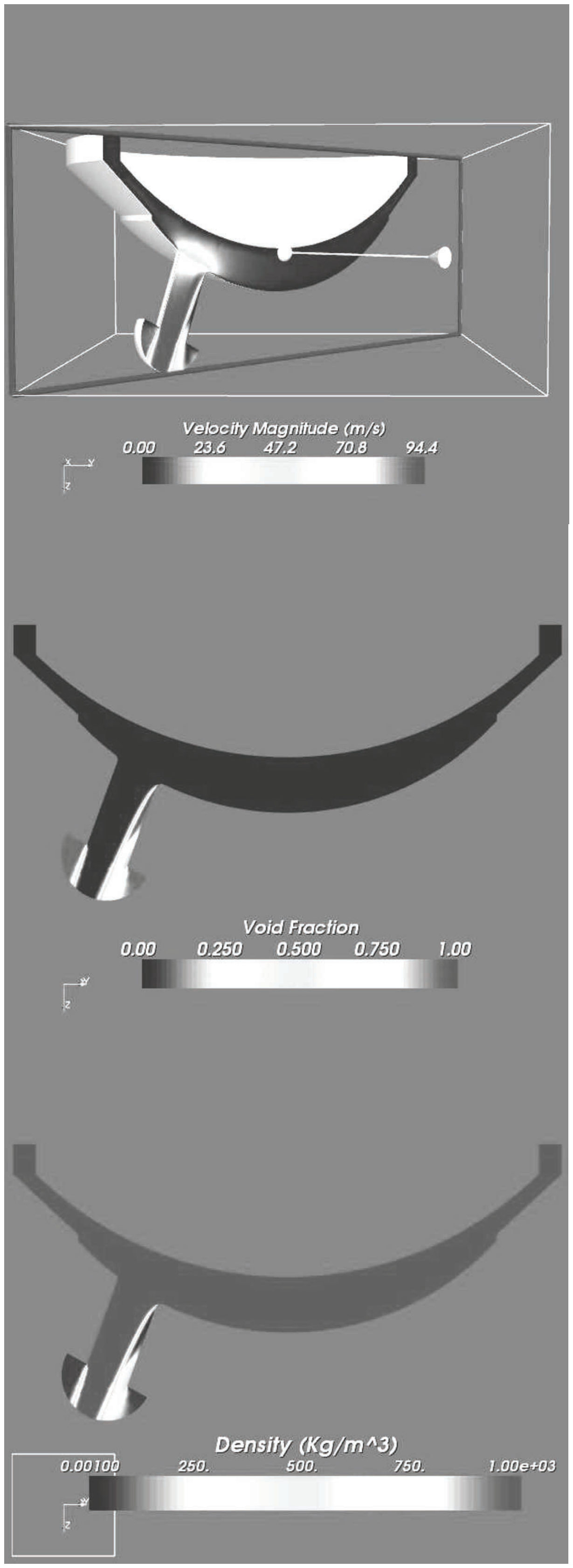


Fig. 1 Flash-boiling in a Diesel injector nozzle. Results by Gopalakrishnan and Schmidt.

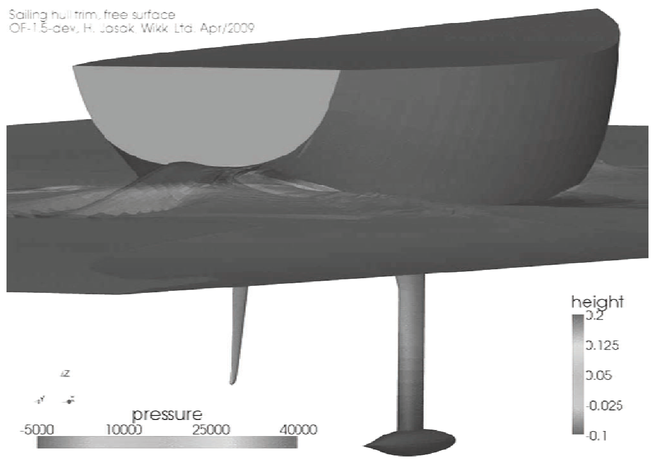
The flash boiling model by Gopalakrishnan and Schmidt can be immediately tested on geometries of industrial interest without re-implementation. An example is given in Fig. 1, on a 3-D geometry of an asymmetric fuel injector nozzle.

Flash boiling is initiated at the inlet edge of the nozzle, indicated by the increase in the vapour fraction, and extends to the outlet plenum. Effect of phase change can also be seen in the velocity field.

Floating Body Simulation: 6-DOF,MovingMesh and Free Surface Flows

The Volume-of-Fluid (VOF) free surface flow solver models the governing equations as a single continuum with a jump in properties at the free surface. The volume fraction variable is used to follow the interface motion and used to calculate the corresponding jump in physical properties (μ, ρ).

The VOF solver in OpenFOAM has been developed in several stages. The first generation (Ubbink and Issa, 1999) uses compressive discretisation on the volume fraction equation, with limitations on cases with dominant surface tension. Subsequent variants use the VOF formulation from a multi-phase flow (Rusche, 2003) with implicit compression terms, with or without compression flux limiting. As an illustration, Fig. 2 shows two examples of free surface flow around partially submerged bodies.


Sailing hull trim, free surface
OF-1.5-dev, H. Jasak, Wikki Ltd. Apr/2009
height
0.2
0.125
0.05
-0.025
-0.1
pressure
-5000
10000
25000
40000

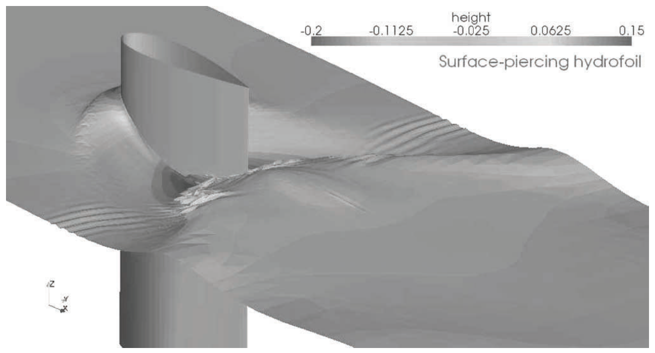
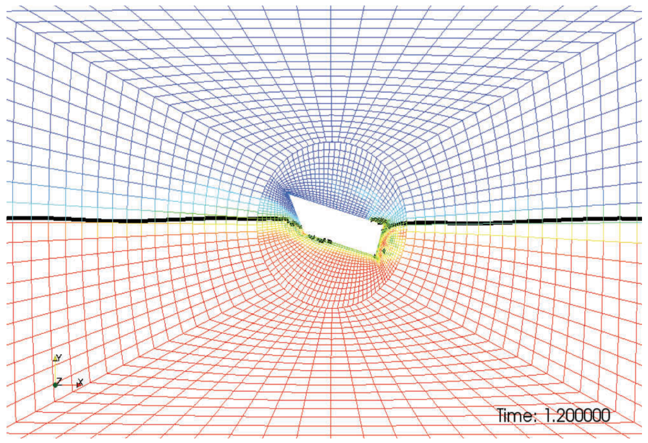


Fig. 2 Examples of free surface flows.

A natural extension of the VOF solver involves floating bodies in free surface flows, combining the VOF flow solver with a 6 Degree of Freedom (6-DOF) solid body motion solver. Forces acting on a solid body are calculated from the flow field and fed to the 6-DOF solver. In return, ordinary differential equations, (ODE) of solid body motion define mesh deformation on the surface of the body.

In terms of implementation, the VOF solver uses the FV discretisation for the flow equations as described above, with support for moving mesh and topological changes. A 6-DOF ODE solver available in the library is used within a `floatingBody` object, calculating the flow forces and motion on the hull patch. A list of floating bodies is held in a `floatingBodyFvMesh`, where an automatic mesh motion solver calculates point motion for the complete mesh, based on prescribed motion on individual boundaries.

For cases of complete capsize, the mesh can be split into two components, coupled with a sliding interface. The inner component undergoes translation and rotation with the body, while the external part undergoes only translation. The two surfaces are coupled using a General Grid Interface (GGI) feature (Beaudoin and Jasak, 2008), originally developed for turbomachinery applications. This is completely encapsulated in the dynamic mesh class, without impact at the toplevel solver.



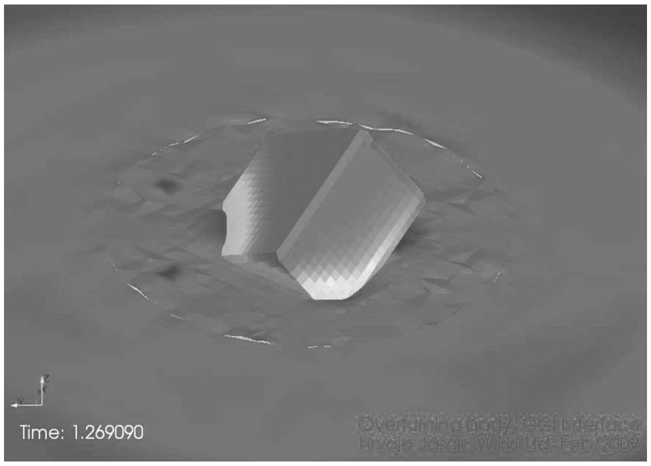

Time: 1.269090
Overturning body, GGI interface
Hrvoje Jasak, Wikki Ltd, Feb 2009

Fig. 3 Simulation of 6-DOF floating bodies with a VOF free surface flow solver.

The separation of tasks between the solver and dynamic mesh class shows the power of object orientation. On one side, flow solver handles the solution of volumetric equations, accounting for a possibility of mesh motion and topological changes. On the other side, a floating body dynamic mesh class executes the motion based on the external forces: in this case, calculated from the free surface flow field. The two are independent from each other: such separation of tasks (flow solver vs. dynamic mesh instance) leads to a clear interface and code re-use.

SUMMARY

This paper describes design principles and basic class layout of OpenFOAM, an object-oriented package for numerical simulation in Continuum Mechanics in C++. On the software engineering side, its advantage over monolithic functional approach is in its modularity and flexibility.

Object orientation breaks the complexity by building individual software components (classes) which group data and functions together and protect the data from accidental corruption. Components are built in families and hierarchies where simpler classes are used to build more complex ones. A toolkit approach implemented in OpenFOAM allows the user to easily and reliably tackle complex physical models in software.

BIBLIOGRAPHY

Schmidt, D. Gopalakrishnan, S. and Jasak, H., (in review) Multidimensional simulation of thermal non equilibrium channel flow. *Journal of Multiphase Flow*.

Ubbink, O. and Issa, R.I., 1999. A method for capturing sharp fluid interfaces on arbitrary meshes. *Journal of Comp. Physics*, 153, pp.26–50.

Rusche, H., 2003. *Computational fluid dynamics of dispersed two-phase flows at high phase fractions*. Ph.D. Imperial College, University of London.

Beaudoin, M. and Jasak, H., 2008. Development of a generalized grid interface for turbomachinery simulations with OpenFOAM. In: o*pen Source CFD International Conference. Berlin*, Germany 4-5 December 2008.